AMENDMENTS TO THE CLAIMS

1. (currently amended) A method of performing dynamic binary ~~translating~~ translation to convert ~~a plurality of basic blocks of~~ subject program code executable on a subject computing architecture into target code executed by a target computing system, comprising the steps of:

grouping together a plurality of basic blocks of the subject program code to form a group block;

decoding the ~~a~~ plurality of basic blocks of the subject program code in the group block;

generating an intermediate representation ~~for each of~~ from said plurality of basic blocks of the subject program code in the group block, wherein the intermediate representation comprises nodes and links arranged as a directed acyclic graph representing expressions, calculations and operations performed by the subject program code, including one or more nodes representing a register definition by the subject program code;

performing a partial dead code elimination optimization on said intermediate representation to generate an optimized intermediate representation, wherein the partial dead code elimination optimization traverses the intermediate representation to create a liveness analysis indicating when the register definition represented by the one or more nodes is a partially dead register definition which is live in one path and dead in another path through the group block;

generating target code from said optimized intermediate representation; and

executing said target code on said target computing system.

2. (currently amended) The method of claim 1, wherein the partial dead code elimination optimization is performed on the one or more of the basic blocks in the group block which end~~ing~~ in non-computed branches or computed jumps.

3. (currently amended) The method of claim 2, wherein the step of performing the partial dead code elimination optimization comprises:

identifying the partially dead register definitions within the one or more ~~a~~ basic blocks in the group block ending in non-computed branches or computed jumps;

~~applying a recursive marking algorithm to~~ marking ~~the~~ child nodes of the one or more nodes relating to the identified partially dead register definitions to produce a set of partially dead nodes; and

performing a code motion optimization algorithm to generate an optimized intermediate representation providing an optimized order for generating target code with reference to the set of partially dead nodes.

4. (currently amended) The method of claim 3, wherein the partially dead register definition identifying step comprises:

for a register definition in ~~a~~ the respective basic block, performing liveness analysis of said register definition in successor basic blocks containing destinations of said non-computed branches or computed jumps; and

identifying said register definition as being partially dead if said register definition is dead in at least one successor basic block and live in at least one other successor basic block.

5. (original) The method of claim 4, further comprising forming a set of identified partially dead register definitions.

6. (currently amended) The method of claim 5, further comprising applying a recursive marking algorithm to identify partially dead child nodes in the intermediate representation of the one or more nodes representing the identified partially dead register definitions.

7. (currently amended) The method of claim 6, wherein said recursive marking algorithm identifies a node in the intermediate representation as a partially dead child node by ensuring that the node is not referenced by either a live node or a node relating to a live register definition.

8. (original) The method of claim 6, wherein said recursive marking algorithm identifies partially dead child nodes as those nodes which are only referenced in the intermediate representation by other partially dead nodes or partially dead register definitions.

9. (original) The method of claim 8, wherein said recursive marking algorithm includes the steps of:

determining a dead count for a child node, wherein the dead count is the number of partially dead nodes referencing the child node in the intermediate representation;

determining a reference count for the child node, wherein the reference count is the number of references to the child node in the intermediate representation; and

identifying a child node as a partially dead when its dead count equals its reference count.

10. (original) The method of claim 6, wherein said recursive marking algorithm further recursively identifies whether the child nodes of identified partially dead child nodes are also partially dead.

11. (currently amended) The method of claim 3, wherein the code motion optimization algorithm comprises:

for each identified partially dead register definition,

determining a path of nodes in the intermediate representation for said partially dead register definition which are live,

discarding the nodes in the intermediate representation for said partially dead register definition which are dead, and

determining partially live paths of nodes in the intermediate representation for said partially dead register definition and moving corresponding nodes into said partially live paths, wherein the nodes in the partially live paths are partially dead nodes, further wherein a partially live path of nodes exists for each respective branch or jump.

12. (original) The method of claim 11, wherein each node in the intermediate representation includes an associated variable which identifies with which partially live path of nodes it is associated.

13. (original) The method of claim 11, wherein said target code generating step comprises:

initially generating target code for all fully live nodes of a partially dead register definition; and

next generating target code for said partially live paths of nodes in the intermediate representation for said partially dead register definition.

14. (original) The method of claim 11, wherein the code motion optimization algorithm further prevents consecutive load and store operations in the intermediate representation from being moved into one of the partially live paths.

15. (original) The method of claim 3, further comprising the step of performing a load-store aliasing optimization.

16. (currently amended) A computer-readable storage medium having software resident thereon in the form of computer-readable code executable by a target computer system to perform the following steps to ~~translate a plurality of basic blocks of program code~~ perform dynamic binary translation to convert subject program code executable on a subject computing architecture into target code executed by a target computing system, the steps comprising:

grouping together a plurality of basic blocks of the subject program code to form a group block;

decoding the~~a~~ plurality of basic blocks of the subject program code in the group block;

generating an intermediate representation from~~for each of~~ said plurality of basic blocks of the subject program code in the group block, wherein the intermediate representation comprises nodes and links arranged as a directed acyclic graph representing expressions, calculations and operations performed by the subject program code, including one or more nodes representing a register definition by the subject program code;

performing a partial dead code elimination optimization on said intermediate representation to generate an optimized intermediate representation, wherein the partial dead code elimination optimisation traverses the intermediate representation to create a liveness analysis indicating when the register definition represented by the one or more nodes is a partially dead register definition which is live in one path and dead in another path through the group block;~~and~~

generating target code from said optimized intermediate representation; and

executing said target code on said target computing system.

17. (currently amended) The computer-readable storage medium of claim 16, wherein the partial dead code elimination optimization is performed on the one or more basic blocks in the group block which end~~ing~~ in non-computed branches or computed jumps.

18. (currently amended) The computer-readable storage medium of claim 17, wherein the step of performing the partial dead code elimination optimization comprises:

identifying the partially dead register definitions within the one or more ~~a~~ basic blocks in a group block ending in non-computed branches or computed jumps;

~~applying a recursive marking algorithm to~~ marking the child nodes of the one or more nodes relating to the identified partially dead register definitions to produce a set of partially dead nodes; and

performing a code motion optimization algorithm to generate an optimized intermediate representation providing an optimized order for generating target code with reference to the set of partially dead nodes.

19. (currently amended) The computer-readable storage medium of claim 18, wherein the partially dead register definition identifying step comprises:

for a register definition in ~~a~~the respective basic block, performing liveness analysis of said register definition in successor basic blocks containing destinations of said non-computed branches or computed jumps; and

identifying said register definition as being partially dead if said register definition is dead in at least one successor basic block and live in at least one other successor basic block.

20. (original) The computer-readable storage medium of claim 19, said computer-readable code further executable for forming a set of identified partially dead register definitions.

21. (currently amended) The computer-readable storage medium of claim 20, said computer-readable code further executable for applying a recursive marking algorithm to identify partially dead child nodes in the intermediate representation of the one or more node representing the identified partially dead register definitions.

22. (currently amended) The computer-readable storage medium of claim 21, wherein said recursive marking algorithm identifies a node in the intermediate representation as a partially dead child node by ensuring that the node is not referenced by either a live node or a node relating to a live register definition.

23. (original) The computer-readable storage medium of claim 22, wherein said recursive marking algorithm identifies partially dead child nodes as those nodes which are only referenced in the intermediate representation by other partially dead nodes or partially dead register definitions.

24. (original) The computer-readable storage medium of claim 23, wherein said recursive marking algorithm includes the steps of:

determining a dead count for a child node, wherein the dead count is the number of partially dead nodes referencing the child node in the intermediate representation;

determining a reference count for the child node, wherein the reference count is the number of references to the child node in the intermediate representation; and

identifying a child node as a partially dead when its dead count equals its reference count.

25. (original) The computer-readable storage medium of claim 21, wherein said recursive marking algorithm further recursively identifies whether the child nodes of identified partially dead child nodes are also partially dead.

26. (currently amended) The computer-readable storage medium of claim 18, wherein the code motion optimization algorithm comprises:

for each identified partially dead register definition,

determining a path of nodes in the intermediate representation for said partially dead register definition which are live,

discarding the nodes in the intermediate representation for said partially dead register definition which are dead, and

determining partially live paths of nodes in the intermediate representation for said partially dead register definition and moving corresponding nodes into

said partially live paths, wherein the nodes in the partially live paths are partially dead nodes, further wherein a partially live path of nodes exists for each respective branch or jump.

27. (original) The computer-readable storage medium of claim 26, wherein each node in the intermediate representation includes an associated variable which identifies with which partially live path of nodes it is associated.

28. (original) The computer-readable storage medium of claim 26, wherein said target code generating step comprises:

initially generating target code for all fully live nodes of a partially dead register definition; and

next generating target code for said partially live paths of nodes in the intermediate representation for said partially dead register definition.

29. (original) The computer-readable storage medium of claim 26, wherein the code motion optimization algorithm further prevents consecutive load and store operations in the intermediate representation from being moved into one of the partially live paths.

30. (original) The computer-readable storage medium of claim 18, further comprising the step of performing a load-store aliasing optimization.

31. (currently amended) ~~In combination~~A computer apparatus comprising:

a ~~target~~ processor;

a memory; and

translator code stored in the memory and executed by the processor to perform dynamic binary translation to convert ~~dynamically convert for translating a plurality of basic blocks of~~ subject program code executable on a subject processor into target code executed by the processor interleaved with execution of the translator code, said translator code comprising code ~~executable~~executed by said ~~target~~ processor for performing the following steps:

grouping together a plurality of basic blocks of the subject program code to form a group block;

decoding the ~~a~~ plurality of basic blocks of the subject program code in the group block;

generating an intermediate representation ~~for each of~~ from said plurality of basic blocks of the subject program code in the group block, wherein the intermediate representation comprises nodes and links arranged as a directed acyclic graph representing expressions, calculations and operations performed by the subject program code, including one or more nodes representing a register definition by the subject program code;

performing a partial dead code elimination optimization on said intermediate representation to generate an optimized intermediate representation, wherein the partial dead code elimination optimisation traverses the intermediate representation to create a liveness analysis indicating when the register definition represented by the one or more nodes is a partially dead register definition which is live in one path and dead in another path through the group block;

~~and~~ generating target code from said optimized intermediate representation; and

executing said target code on the processor.

32. (currently amended) The ~~combination~~computer apparatus of claim 31, wherein the partial dead code elimination optimization is performed on the one or more basic blocks in the group block which end~~ing~~ in non-computed branches or computed jumps.

33. (currently amended) The ~~combination~~computer apparatus of claim 32, wherein the step of performing the partial dead code elimination optimization comprises:

identifying the partially dead register definitions within the one or more ~~a~~ basic blocks in the group block ending in non-computed branches or computed jumps;

~~applying a recursive marking algorithm to~~ marking the child nodes of the one or more nodes relating to the identified partially dead register definitions to produce a set of partially dead nodes; and

performing a code motion optimization algorithm to generate an optimized intermediate representation providing an optimized order for generating target code with reference to the set of partially dead nodes.

34. (currently amended) The ~~combination~~computer apparatus of claim 33, wherein the partially dead register definition identifying step comprises:

for a register definition in ~~a~~ the respective basic block, performing liveness analysis of said register definition in successor basic blocks containing destinations of said non-computed branches or computed jumps; and

identifying said register definition as being partially dead if said register definition is dead in at least one successor basic block and live in at least one other successor basic block.

35. (currently amended) The ~~combination~~computer apparatus of claim 34, said translator code further comprising code executable by said target processor for forming a set of identified partially dead register definitions.

36. (currently amended) The ~~combination~~computer apparatus of claim 35, said translator code further comprising code executable by said target processor for applying a recursive marking algorithm to identify partially dead child nodes in the intermediate representation of the one or more nodes representing the identified partially dead register definitions.

37. (currently amended) The ~~combination~~computer apparatus of claim 36, wherein said recursive marking algorithm identifies a node in the intermediate representation as a partially dead child node by ensuring that the node is not referenced by either a live node or a node relating to a live register definition.

38. (currently amended) The ~~combination~~computer apparatus of claim 36, wherein said recursive marking algorithm identifies partially dead child nodes as those nodes which are only referenced in the intermediate representation by other partially dead nodes or partially dead register definitions.

39. (currently amended) The ~~combination~~computer apparatus of claim 38, wherein said recursive marking algorithm includes the steps of:

determining a dead count for a child node, wherein the dead count is the number of partially dead nodes referencing the child node in the intermediate representation;

determining a reference count for the child node, wherein the reference count is the number of references to the child node in the intermediate representation; and

identifying a child node as a partially dead when its dead count equals its reference count.

40. (currently amended) The ~~combination~~computer apparatus of claim 36, wherein said recursive marking algorithm further recursively identifies whether the child nodes of identified partially dead child nodes are also partially dead.

41. (currently amended) The ~~combination~~computer apparatus of claim 33, wherein the code motion optimization algorithm comprises:

for each identified partially dead register definition,

determining a path of nodes in the intermediate representation for said partially dead register definition which are live,

discarding the nodes in the intermediate representation for said partially dead register definition which are dead, and

determining partially live paths of nodes in the intermediate representation for said partially dead register definition and moving corresponding nodes into said partially live paths, wherein the nodes in the partially live paths are partially dead nodes, further wherein a partially live path of nodes exists for each respective branch or jump.

42. (currently amended) The ~~combination~~computer apparatus of claim 41, wherein each node in the intermediate representation includes an associated variable which identifies with which partially live path of nodes it is associated.

43. (currently amended) The ~~combination~~computer apparatus of claim 41, wherein said target code generating step comprises:

initially generating target code for all fully live nodes of a partially dead register definition; and

next generating target code for said partially live paths of nodes in the intermediate representation for said partially dead register definition.

44. (currently amended) The ~~combination~~computer apparatus of claim 41, wherein the code motion optimization algorithm further prevents consecutive load and store operations in the intermediate representation from being moved into one of the partially live paths.

45. (currently amended) The ~~combination~~computer apparatus of claim 33, said translator code further comprising code executable by said target processor for performing a load-store aliasing optimization.